EDN | COLUMNIST

Simulation in fuzzy-controller design

DAVID BRUBAKER,
FUZZY-LOGIC
CONTRIBUTING EDITOR
brubaker@cup.portal.com

In my seven-year career as a basketball player—seventh grade through freshman year in college—the most fun I had was during the last eight weeks of the last year. The freshman coach suggested it would be good for me to get varsity experience and invited me to join the "Red Team," those players who studied films of opposing teams and attempted to emulate those teams in practice.

This was great fun, working hard with the rest of the Red Team to put together both the offenses and defenses of the other PAC-8 teams. Of these teams we tried to be, the only one I clearly remember is UCLA. That was Lew Alcindor's (now Kareem Abdul-Jabbar's) sophomore year, and with him as center, the Bruins went on to win the NCAA championship that year, and the next two years as well.

What does this have to do with fuzzy logic—beyond our Red Team being in the set of UCLA-like basketball teams with degree of membership less than or equal to 0.01? The stretch is actually not that great. In emulating other teams, the Red Team was a simulation. Varsity could not practice against the actual teams they would face, but they needed to prepare. We Red Team members did our best to simulate the other teams, and, overall, I believe we helped.

Simulation also plays an important role when designing fuzzy systems, especially fuzzy controllers. For a number of reasons, a designer often cannot connect the controller being developed directly to the controlled system until late in the development process. In early design stages, he or she must instead use a simulation environment. Just as the varsity coach used the Red Team's simulation of Stanford's opponents to design and refine plays, options, and defenses, the fuzzy-system designer uses a simulation of the controlled system to design and tune the fuzzy controller.

There is another similarity. My Red Team was a poor simulation of the UCLA Bruins. Similarly, in system design, a good simulation model is often difficult to create; recall that fuzzy systems are often the best solution when an analytic model of the controlled system is unavailable. The easy justification, the one we used as the Red Team, is that a poorly representative model is better than no model at all.

In fuzzy-system design, there is deeper, more significant justification. Because of the robustness of fuzzy systems, their insensitivity to variations in the parameters of the systems they control, a poor-quality model can be extremely helpful in the design. Designers have noticed that they often do not need to perform final tuning of fuzzy parameters when interacting with the actual target system, even when the simulation model used during design inadequately represents the system it is modeling. This is an often overlooked feature of fuzzy systems.

There is a purpose for this lengthy introduction. This month we will discuss the use of a simulator in designing the rulebase of a fuzzy system. Our model is simple, and the simulation therefore accurate. The model is of the series RLC circuit introduced last time (*EDN*, January 5, 1994, pg 167).

Recall that we are designing a fuzzy PD+I controller that charges a 100-μF capacitor through a 10-mH inductor (**Fig 1**). We'll design the PD part of the system first, concentrating on meeting rise time and overshoot requirements, less than 1 msec and less than 5%, respectively. Once satisfied with the PD response, we'll implement the I (integral) component separately, using it to minimize steady-state error.

We start with the membership functions. Defining them is the simplest part of the design. As discussed last time, we will work with normalized domains, deferring dealing with real-world ranges until later.

For both the error value, *error*, (the P input) and its derivative, *error_dot*, (the D input), we define seven fuzzy values, and therefore seven membership functions (**Fig 2a**). The number seven is somewhat arbitrary, although most simple to moderately complex systems use between three and seven fuzzy values for each input.

For function shapes, we choose broader membership functions for the regions removed from zero where we want coarse control, and narrower functions for the regions close to zero where we need fine control. Although any appropriate function representation is possible, we'll use triangles and trapezoids, which are easier to represent and to work with.

The PD output is the V_{IN} applied to the RLC circuit. Again, with normalized domain, we will use seven singletons as the possible actions, spaced evenly between ±1, inclusive (**Fig 2b**). A singleton is a single, scalar output value. It is not a fuzzy set, represented with a membership function, but is rather a single, crisp value. In a future column, we will discuss types of defuzzification and compare fuzzy output sets combined and defuzzified using center of mass to singletons combined using a weighted average. The two achieve quite similar results, and in control, where execution speed is important, I will most often use singletons.

If creating membership functions is the easiest design step, identifying rules is the most complicated (**Fig 3**). We will now start using the simulator. The "design" process consists of repeatedly resetting the simulator to its initial conditions and

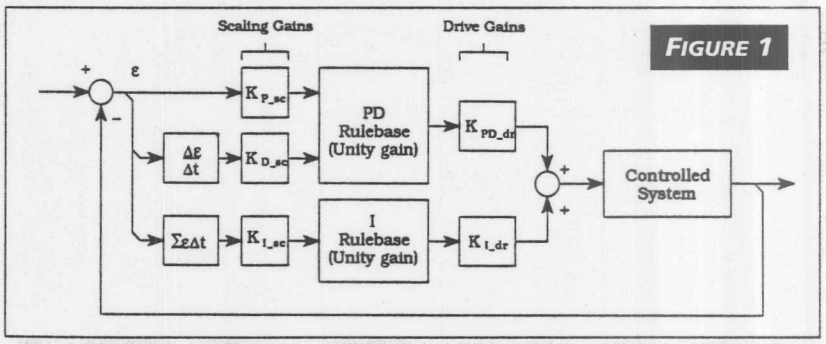


FIGURE 1

A fuzzy PD+I controller charges a 100-μF capacitor through a 10-mH inductor.

PHOTOGRAPHY / LIANE ENKELIS

single-stepping through a commanded input voltage step. Each time the system enters a region in the input space with an as-yet-undefined action, the designer determines an appropriate output action, inserts this into the rule specified by the given input space, and single-steps the simulator into the region. If the selected action is correct, system operation will be as desired. If not, the designer specifies a different action and restarts the simulation. The designer repeats this sequence of restarting, single-stepping, and testing numerous times until he or she has defined the entire rulebase.

Let's see how this works. From experience, we know that bang-bang control can effectively move a system close to its setpoint, after which some form of continuous control achieves and maintains the setpoint. This is a traditional method of using highly undamped control far from the setpoint and allowing it to give way to a damped control near the setpoint. This is the approach we shall take with the fuzzy controller.

Fig 3a shows the desired operation of the system. When the actual and commanded capacitor values are equal, both *error* and *error_dot* are zero—the system resides in cell A. If the command is for a positive voltage step, the system jumps horizontally to B. As the capacitor starts to charge, the derivative goes negative until *error* is PL (Positive_Large) and *error_dot* is NL (Negative_Large)—the upper right corner of the matrix.

The capacitor continues to charge, and as *error* becomes smaller, the system enters region C. If the designer has correctly assigned rule actions, the system will quickly spiral through regions C and D and back toward cell A. This is the setpoint.

The heavy black line below region D is a boundary the response cannot pass, because the unshaded cells in Fig 3a must be the mirror image of the shaded cells to respond to a negative step command; that is, they must have identical amplitude but opposite polarity actions. If the system response crosses the black line during the last stages of a positive step, it will act as if it were in the middle stages of a negative step response.

Fig 3b shows the regions in the order that rules are assigned. Assume the system is at rest with the capacitor fully discharged and with no current flowing—it is in cell 1 at the center of the matrix where *error* and *error_dot* are both ZE (zero). Using the simulator in single-step mode, the designer commands the system to step to 5V. The system jumps laterally into region 2, to the cell "*error* is PL AND *error_dot* is ZE."

We want a maximum positive output where there is a large, positive error; this is the first "bang" of the bang-bang controller. We therefore assign PL (Positive_Large) as the action for all cells in region 2. For example, the rule for the center cell in the column is

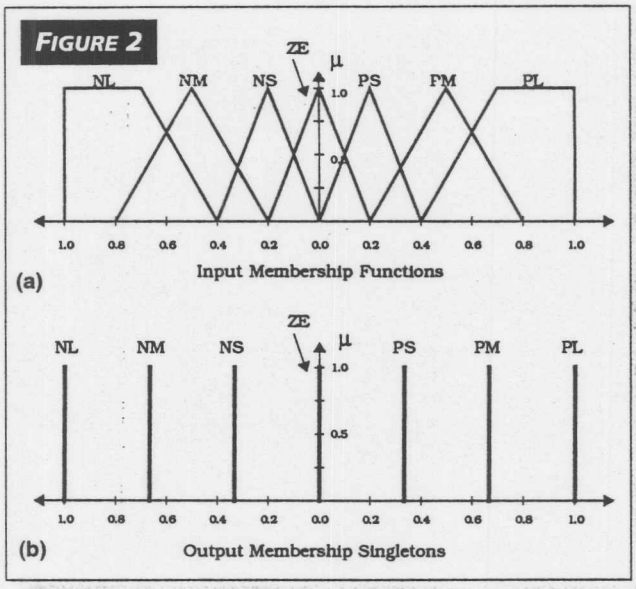
IF *error* is PL AND *error_dot* is ZE THEN V_{IN} is PL;

Using the simulator, we check to see if the response is what we expected. It is. In response to the large positive drive, the simulated capacitor starts to charge, and after several Δt increments, the derivative term decreases from zero to Negative_Large (NL). The system is now in the upper right cell in the matrix.

As the error decreases further, the system enters the second region of the phase plane, the first cell being where "*error* is PM AND *error_dot* is NL." We want this region, labeled 3 in Fig 3b, to provide the second bang—to put on the brakes. We therefore set all cells in this region to NL.

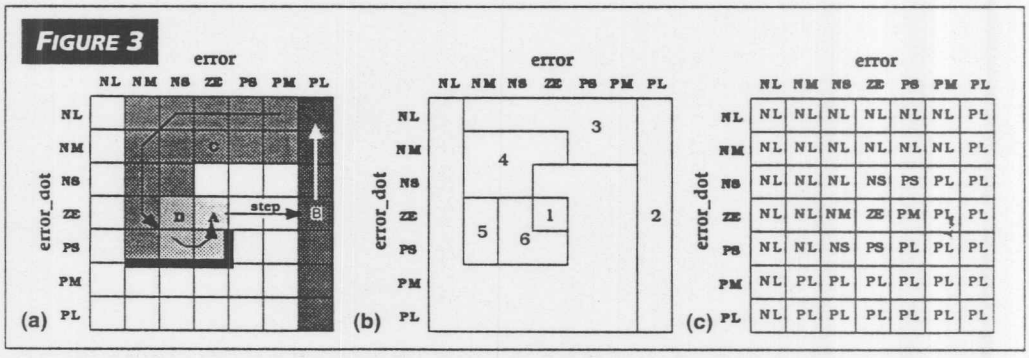
As we continue to step the simulator, the response continues toward the left and across the top of the rule matrix, where *error* is decreasing but *error_dot* remains maximum. Even as it moves into and through regions 4 and 5 of Fig 3b, we continue to require the large

FIGURE 2



The plot in (a) shows the input membership functions used for both the P and D inputs. Both domains range between ±1.0, with appropriate scaling performed by scaling gains. Input membership functions are standard, with functions nearer to zero being somewhat more narrow than functions farther from zero. The plot in (b) shows the output singletons used for the PD drive, V_{IN}.

FIGURE 3



These three matrices are of the rulebase that is used for the PD component of the controller. The matrix in (a) shows the desired control actions. The figure (b) shows the sequence taken to design the rulebase, and (c) shows the final set of rules.

negative output, NL. Only as the response moves into region 6 do we terminate bang-bang control and allow less than maximum outputs to dampen the system in toward the setpoint. By continuing to step the simulator through its sequence, the rules for the three cells in Region 6 become

IF *error* is NS AND *error_dot* is ZE
THEN V_{IN} is NM;
IF *error* is NS AND *error_dot* is PS
THEN V_{IN} is NS;
IF *error* is ZE AND *error_dot* is PS
THEN V_{IN} is PS;

Fig 3c shows the completed rule matrix, with the actions for a negative commanded step added as well. Note that the rulebase shown in Fig 3c is for the full PD+I implementation. For the PD-only system, I used a rule action of PL (Positive_Large) for the rule with condition error is ZE (zero) and error_dot is PS (Positive_Small), intending to minimize as much as possible the offset error. When we add integral control, we can reduce the PL action to PS.

With the exception of this single rule (and its negative step counterpart), the rulebase existed in its final form after the first design cut. The total rule design process took approximately two hours and involved single-stepping and running numerous simulations while identifying and modifying rules for the various cells. The membership functions did not change from their initial shapes and positions.

However, I definitely "played" with the scaling gains as part of the tuning process. As shown in Fig 1, the PD part of the controller has three gains associated with it: scaling gains for the P and D components, and a drive gain for the combined PD output. We first set scaling gains to allow maximum error values to fully exercise the normalized input domains, and we'll subsequently modify them as part of system tuning.

Setting the initial gain for the P term is straightforward. Recall that we normalize values into a ±1.0 range. The maximum possible error for the P term is 5V, and we therefore use a scaling gain of 0.2. To set the D scaling gain, we could calculate the maximum anticipated charging current and thereby the maximum anticipated dV/dt. However, it is simpler to use the simulator to apply full drive to the RLC circuit and watch the capacitor charge to one-half the maximum step size. Doing so, we observe a maximum ΔV of approximately 0.4V per sample interval Δt and therefore set the initial D scaling gain to 2.5. Incidentally, in the simulation, we multiply $\Delta V/\Delta t$ by Δt to have the derivative input in terms of volts.

We also select the PD drive gain empirically, with an initial value of 20 to provide sufficient drive in the first half of the step response to keep the drive voltage to the RLC circuit saturated. A maximum rulebase output of +1 would result in an applied V_{IN} of 20V, limited by the physical system to +15V.

In the next column, we'll complete the design by building the I (integral) leg of the controller.

VOTE

Please use the Information Retrieval Service card to rate this article (circle one):

High Interest	Medium Interest	Low Interest
582	583	584